

华中科技大学

计算机视觉课程实验报告

实验四：卷积神经网络可解释性分析

姓 名：	李嘉鹏
学 院：	计算机科学与技术学院
专 业：	数据科学与大数据技术
班 级：	大数据 2101 班
学 号：	U202115652
指导教师：	刘康

2023 年 12 月 30 日

目 录

实验四 卷积神经网络可解释性分析	1
1.1 实验简介	1
1.1.1 实验背景	1
1.1.2 实验目的	2
1.2 实验设计与步骤	3
1.2.1 全局平均池化（GAP）介绍	3
1.2.2 Grad-CAM 介绍	3
1.2.3 Layer-CAM 介绍	8
1.2.4 可解释性分析	8
1.2.5 参数选择	9
1.3 实验结果	10
1.3.1 第一组实验结果	10
1.3.2 第二组实验结果	14
1.3.3 第三组实验结果	18
1.4 结果分析	19
1.4.1 总体结果分析	19
1.4.2 全局平均池化层和最后一个卷积层特征图效果的对比	19
1.4.3 Grad-CAM 和 Layer-CAM 效果的对比	19
1.5 参考文献	21
1.6 源代码	21
1.6.1 gradcam_avgpool.py	21
1.6.2 gradcam_lastconv.py	24
1.6.3 layercam.py	26

实验四 卷积神经网络可解释性分析

1.1 实验简介

1.1.1 实验背景

神经网络的可解释性是神经网络内部运行的方式、原因和结构是可以在逻辑上被理解的。尽管神经网络在许多领域取得了重大突破，但它们通常被认为是黑盒模型，也就是说我们很难解释其内部的决策过程。

- 复杂性：神经网络由许多层和节点组成，它们的参数达到了巨大的量级，并且极为复杂。因此人类难以理解神经网络在每个层级和节点上的具体功能和贡献。
- 非线性：神经网络通过激活函数引入非线性，导致神经网络的决策过程更复杂。
- 特征提取：神经网络能够从原始数据中提取特征，但是它提取的特征在人类看来很大概率是“无意义”或者“不知道意义何在”的，我们自然也难以理解网络是如何识别和利用这些特征来做出决策的。

可见神经网络的决策过程在许多场景下难以直观解释，近年来已经出现了一些方法来增强神经网络的解释性。

本实验主要是基于类激活热力图（CAM）的可视化，也就是将神经网络的训练过程进行可视化，从而对其理解更加深入。图 1 展示了图像分类任务在多种 CAM 算法下的热力图^[1]，图中色块的颜色越红，说明神经网络对该块的关注度越大。

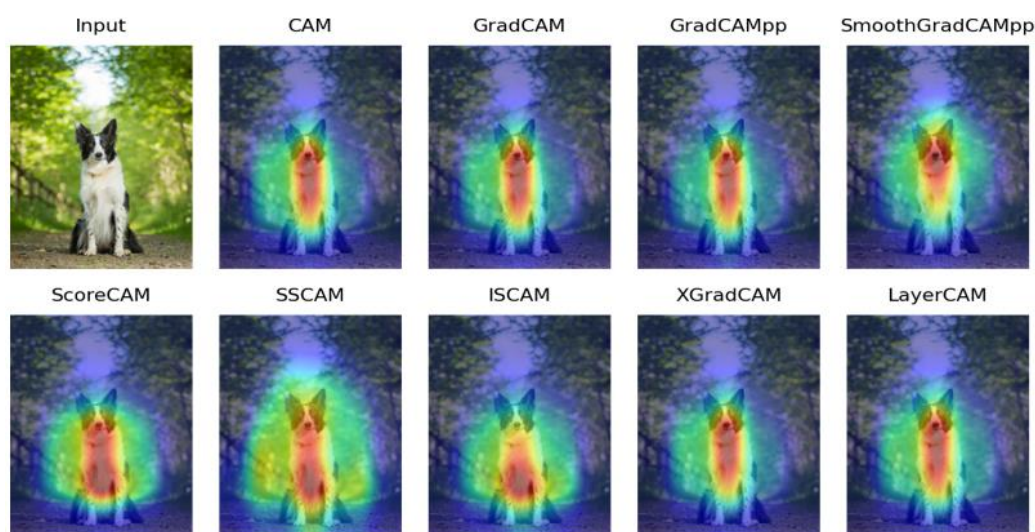


图 1：图像分类任务在多种 CAM 算法下的热力图

1.1.2 实验目的

本实验要求针对已训练好的卷积神经网络，给定一张输入图片，生成该图片对于特定类别的可解释性分析结果。实验将提供基于 PyTorch 和 TensorFlow 的两个版本的二分类模型，该模型可用于猫和狗的分类(class 0 为猫、class 1 为狗)，其中 PyTorch 使用 AlexNet 网络架构，TensorFlow 使用 VGG16 架构，任选一个模型即可。对于提供的三张输入图片，分别针对猫和狗的类别进行 Grad-CAM 和 Layer-CAM 的可解释性分析。

需要给出每张输入图片在最后一层卷积层输出的可视化结果（对输出特征图的每一个通道进行可视化），以及每张输入图片分别针对猫和狗两个类别的可解释性分析结果（Grad-CAM 及 Layer-CAM）。

1.2 实验设计与步骤

1.2.1 全局平均池化（GAP）介绍

在常见的卷积神经网络中，全连接层前的卷积层会对图像进行特征提取。在获取特征后，传统的方法是接上全连接层后再进行激活分类，而全局平均池化^[1]（GAP）的思路是使用全局平均池化层代替该全连接层（使用池化层的方式降维），在实际应用中效果提升较明显。

如图 2 右半边所示，GAP 使用一个标量来间接代表最后一层全卷积层的一个 Channel，具体的做法是将每个通道的二维图像做平均，最后每个通道对应一个均值。这一方式大大减少了参数量。

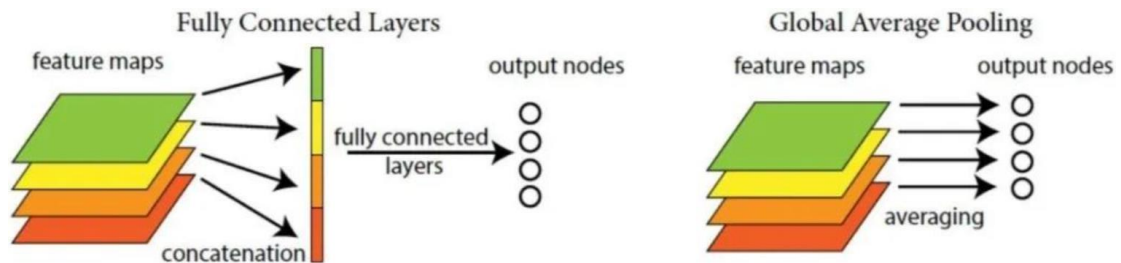


图 2：全局平均池化（GAP）介绍

1.2.2 Grad-CAM 介绍

Grad-CAM^[2]的总体概况如图 3 所示，它可以在图像分类、图像描述生成、视觉问题回答等多个领域得到应用。

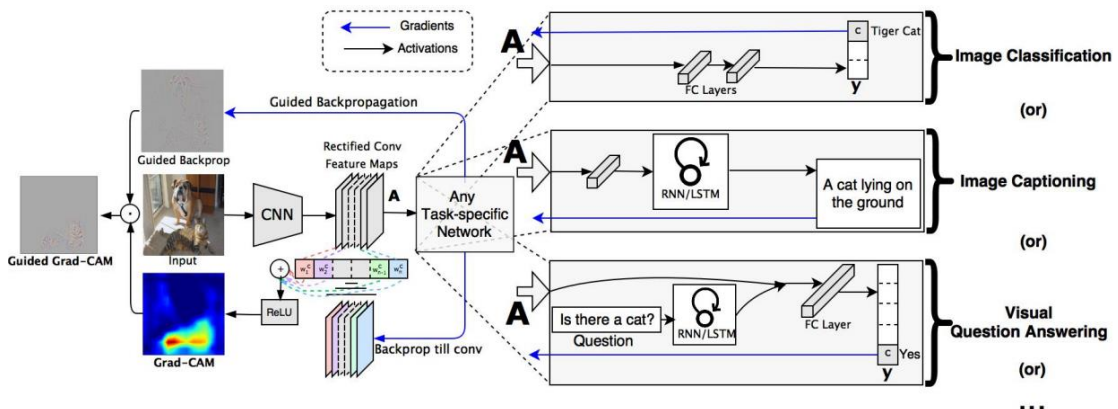


图 3：Grad-CAM 总体概况

Grad-CAM 的算法流程（图 3 的左半部分）如图 4 所示。

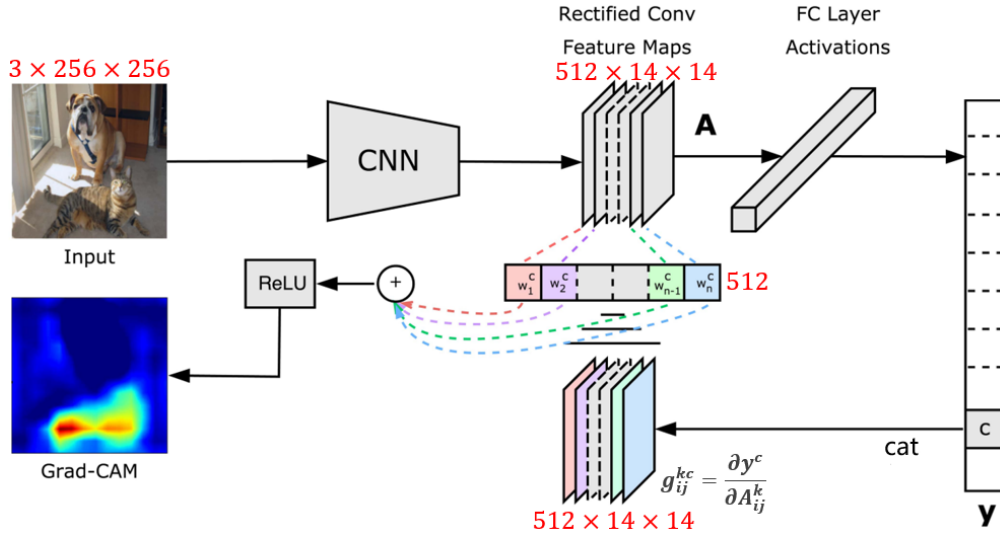


图 4: Grad-CAM 算法流程

由上图可知，在分类模型中，最后一层卷积为矩阵 A ，通道 k 中 (i, j) 的位置被定义为 A_{ij}^k 。最后一层卷积经过全连接层后得到各分类的分数（送入 softmax 前），其中分类 c 的分数为 y^c 。

Grad-CAM 接下来求得分数 y^c 对最后一层卷积 A_{ij}^k 每一个元素的偏导数

$g_{ij}^{kc} = \frac{\partial y^c}{\partial A_{ij}^k}$ ，经过全局平均池化（GAP）得到权重 $w_k^c = \frac{1}{N} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$ （其中 N 为

A_{ij}^k 的像素数），与最后一层卷积的特征图进行加权求和后经过 ReLU 激活函数，最后再经过上采样（双线性插值）到与原始图像相同尺寸，可以得到热力图：

$$L_{\text{Grad-CAM}}^c = \text{ReLU} \left(\underbrace{\sum_k \alpha_k^c A^k}_{\text{linear combination}} \right)$$

综合上面的分析，Grad-CAM 算法的可视化流程分为五个步骤：

- （1）输入一张图像和一个感兴趣的类别（例如 cat）；
- （2）通过模型的 CNN 部分前向传播，得到各类别分数 y （softmax 层之前），进行求导、全局平均池化后得到权重；
- （3）保留选定类别的梯度，忽略其它类别的梯度；
- （4）将选定类别的分数 y^c 反向传播至卷积特征图，组合计算得到粗糙的

CAM 梯度热力图；

(5) 将热力图与反向传播的结果进行点乘，得到高分辨率的 Grad-CAM 可视化图。

其中 y^c 可以进行替换，因此 Grad-CAM 可以分析中间的卷积层。

对于本实验提供的 AlexNet 模型，通过下面的语句输出其结构并观察：

```
print(model.__dict__)
for name, module in model.named_modules():
    print(name, module)
```

部分结果如下：

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
```

```

(5): ReLU(inplace=True)
(6): Linear(in_features=4096, out_features=2, bias=True)
)
)

```

可以看出模型主要由三个模块组成，分别是 features、avgpool 和 classifier。其中 features 的第 10 层为最后一个卷积层，特征图大小为 256*13*13；avgpool 为全局平均池化层，特征图大小为 256*6*6。

下面将解释 Grad-CAM 核心算法的具体实现。在前向传播过程中，features 模块保存对应的梯度信息，avgpool 模块通过对通道进行平均处理得出最后的特征 feature。Grad-CAM 类的代码如下所示。

```

class GradCam:
    def __init__(self, model):
        self.model = model.eval()
        self.feature = None
        self.gradient = None
        self.adaptiveavgpool_features = []

    def save_gradient(self, grad):
        self.gradient = grad

    def save_adaptiveavgpool_features(self, feature):
        self.adaptiveavgpool_features.append(feature)

    def __call__(self, x, interested_class):
        image_size = (x.size(-1), x.size(-2))
        datas = Variable(x)
        heat_maps = []
        for i in range(datas.size(0)):
            .....
            feature = datas[i].unsqueeze(0)
            for name, module in self.model.named_children():
                if name == 'classifier':
                    feature = feature.view(feature.size(0), -1)
                    feature = module(feature)
                if name == 'features':
                    feature.register_hook(self.save_gradient)
                    self.feature = feature
                elif name == 'avgpool':
                    self.save_adaptiveavgpool_features(feature) # 第12层
                    平均池化
            classes = F.softmax(feature)

```



```
.....  
return heat_maps, self.adaptiveavgpool_features
```

为了针对某一类别进行可解释性分析，我引入了一个变量 **interested_class** 记录要分析的类别，将该类的概率设为 **1**，其余类别的概率设为 **0**，构造一个 **one-hot** 向量。然后将模型的梯度清零，反向传播计算模型输出相对于这一 **one-hot** 向量的梯度，并使用全局平均池化（GAP）计算该类的权重 **weight**。接下来，将权重和特征相乘并经过 ReLU 激活函数得到感兴趣区域的 CAM，将其归一化后叠加到原始图像上，生成可视化的热力图。

```
predicted_class = interested_class  
one_hot = torch.zeros_like(classes)  
one_hot[:, predicted_class] = 1  
  
self.model.zero_grad()  
classes.backward(gradient=one_hot, retain_graph=True)  
  
weight = self.gradient.mean(dim=-1, keepdim=True).mean(dim=-2,  
                        keepdim=True)
```

最后，循环遍历平均池化层输出的特征，并将特征图作为一个 16*16 的子图输出。

```
heat_map, adaptiveavgpool_features = grad_cam(test_image,  
                        interested_class)  
  
fig, axs = plt.subplots(16, 16, figsize=(16, 16))  
count = 0  
  
for i in range(16):  
    for j in range(16):  
        feature_map =  
adaptiveavgpool_features[0][0][count].detach().cpu().numpy()  
        axs[i, j].imshow(feature_map, cmap='jet')  
        axs[i, j].axis('off')  
        count += 1  
        if count >= len(adaptiveavgpool_features[0][0]):  
            break  
  
plt.tight_layout()  
plt.savefig(FEATURE_SAVE_NAME, bbox_inches='tight', pad_inches=0)
```

上面展示了依据平均池化层来绘制特征图的代码。如果要依据最后一个卷积

层来绘制特征图，只需要在该卷积层输出特征并忽略平均池化层即可，为此可以设置一个计数器 `round` 来统计当前的层数，代码如下所示。

```
round = 0
for name, module in self.model.named_children():
    if name == 'classifier':
        feature = feature.view(feature.size(0), -1)
    if name == 'features':
        for submodule in module:
            feature = submodule(feature)
            round += 1
            if round == 11:
                self.feature = feature # 最后一个卷积层
feature.register_hook(self.save_gradient)
```

1.2.3 Layer-CAM 介绍

Grad-CAM 的缺点有二：深层生成的粗粒度热力图和浅层生成的细粒度热力图都不够精确；虽然 Grad-CAM 可以分析中间的卷积层，但浅层的分析效果很差。

在 Grad-CAM 的基础上，Layer-CAM^[3]没有使用全局平均池化，而是对特征图的梯度进行了元素级的乘法，从而生成更精细的类别激活图。实际上这一算法相对 Grad-CAM 变化不大，区别只在于去掉了平均池化层，并且使用了下面这句代码完成了梯度和特征之间一一对应的乘法，从而计算权重 `weight`。

```
weight = self.gradient * self.feature
```

输出 `self.gradient` 和 `self.feature` 的形状，如图 5 所示，二者是一致的。

```
self.feature.shape: torch.Size([1, 256, 13, 13])
self.gradient.shape: torch.Size([1, 256, 13, 13])
```

```
Process finished with exit code 0
```

图 5：特征和梯度的形状

1.2.4 可解释性分析

对于每张图片，会针对不同的类别生成最后一层输出的特征图和可解释性热力图。理论上 **Grad-CAM** 和 **Layer-CAM** 的特征图是相同的（因为对于同一模型，其处理同一张图片在同一层的输出必然是相同的），而可解释性热力图应该是大致相同的（通过阅读相关论文发现这两种 CAM 算法在大量测试集上最终输出的图片效果相似）。无论是哪种 CAM 算法，热力图中的红色部分应该聚焦在

识别出来的对象上。

1.2.5 参数选择

为了对比不同层特征图对最后产生的热力图的影响，我在 Grad-CAM 的实验中分别取了最后一个卷积层和全局平均池化层的特征图，并分别生成热力图，最后观察其效果。

不同组的实验条件如表 1 所示。

表 1：不同的可解释性算法和特征图选取策略

组	可解释性算法	特征图选取	特征图大小
1	Grad-CAM	全局平均池化层	256*6*6
2		最后一个卷积层	256*13*13
3	Layer-CAM	最后一个卷积层	256*13*13

1.3 实验结果

1.3.1 第一组实验结果

在 Grad-CAM 选取全局平均池化层的条件下，三张输入图片的输出特征图分别如图 6~图 8 所示。对于猫和狗的分类，热力图分别如图 9~图 14 所示。

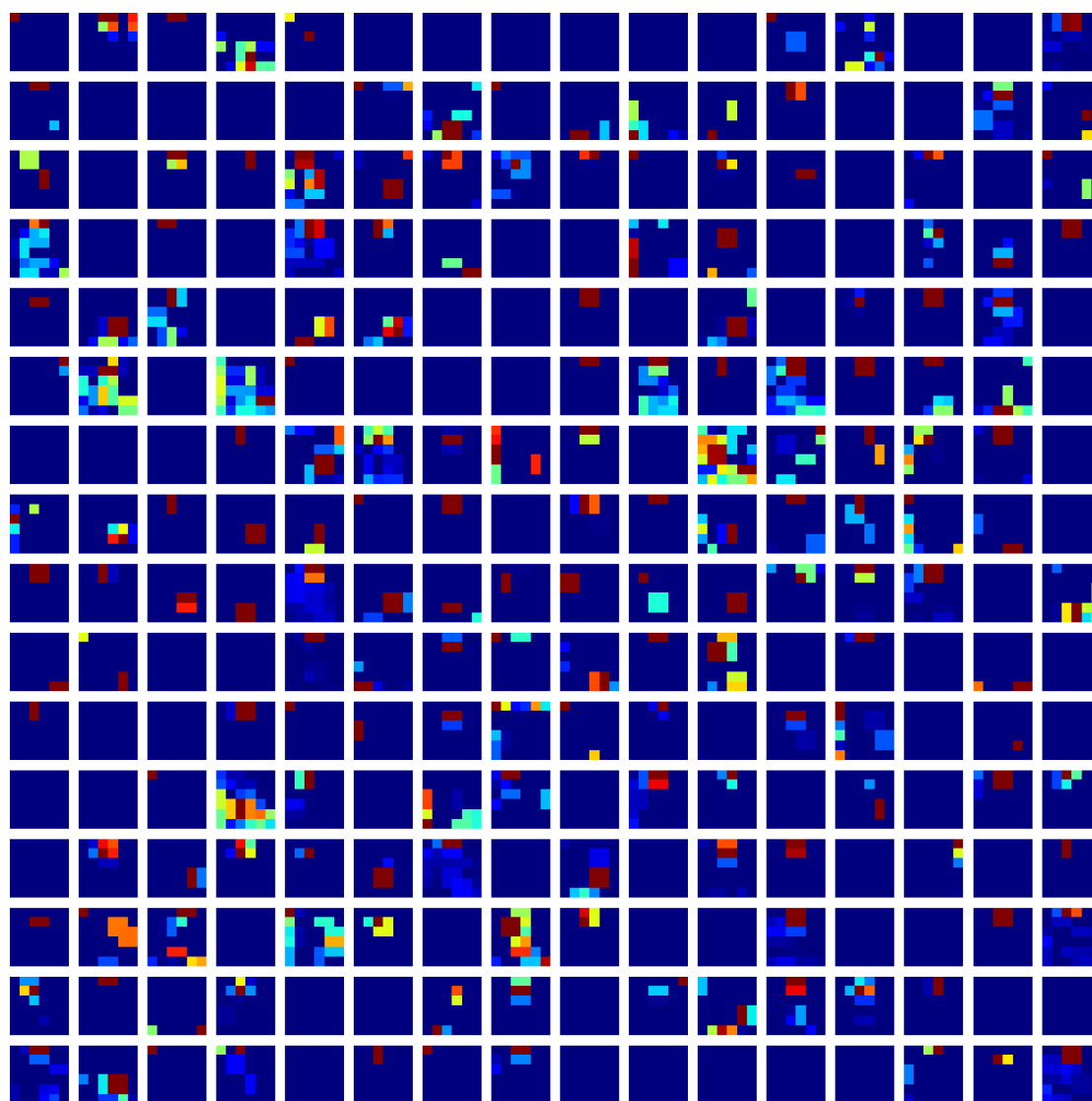


图 6: GradCAM_both_feature_avgpool

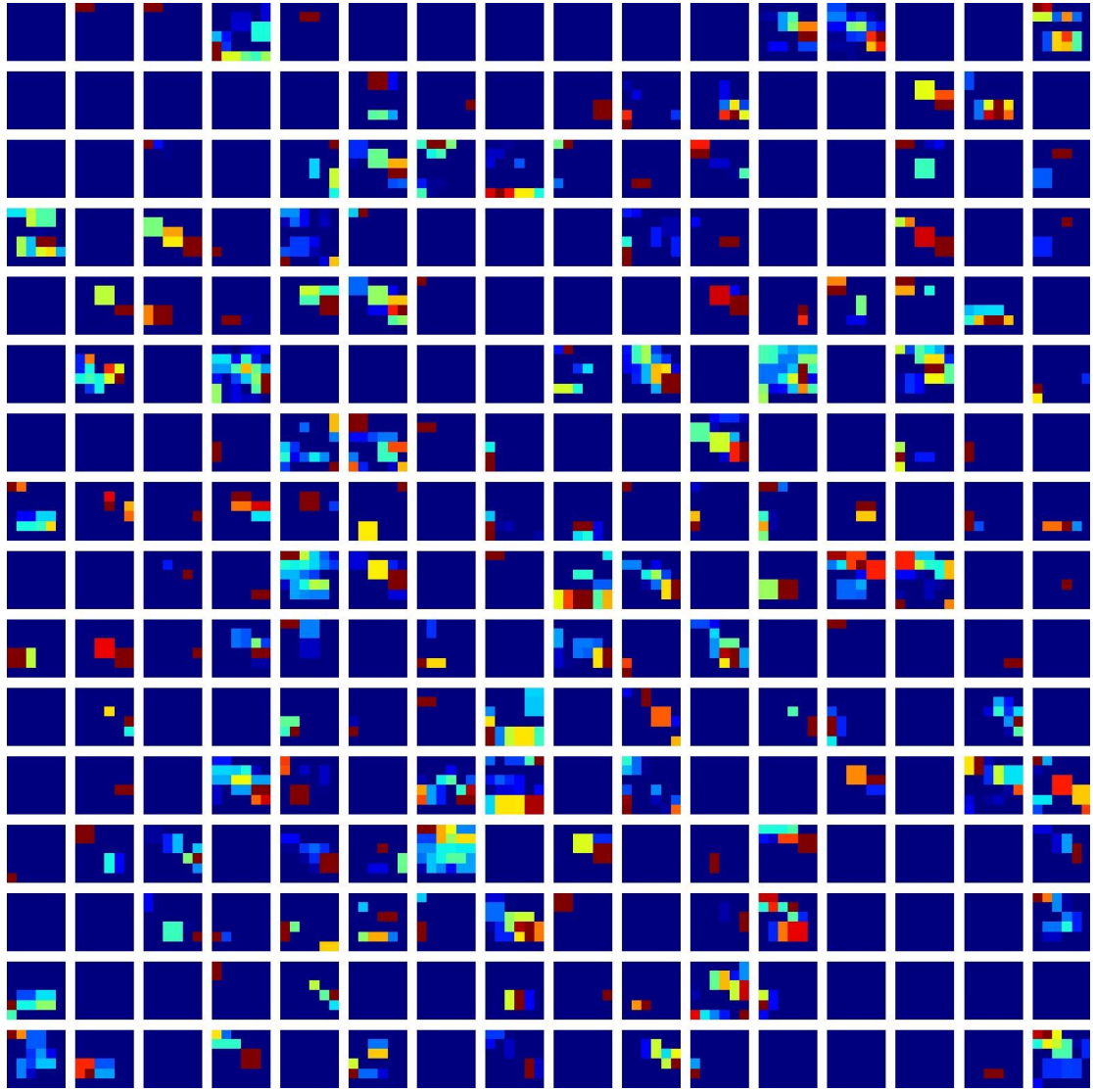


图 7: GradCAM_cat_feature_avgpool

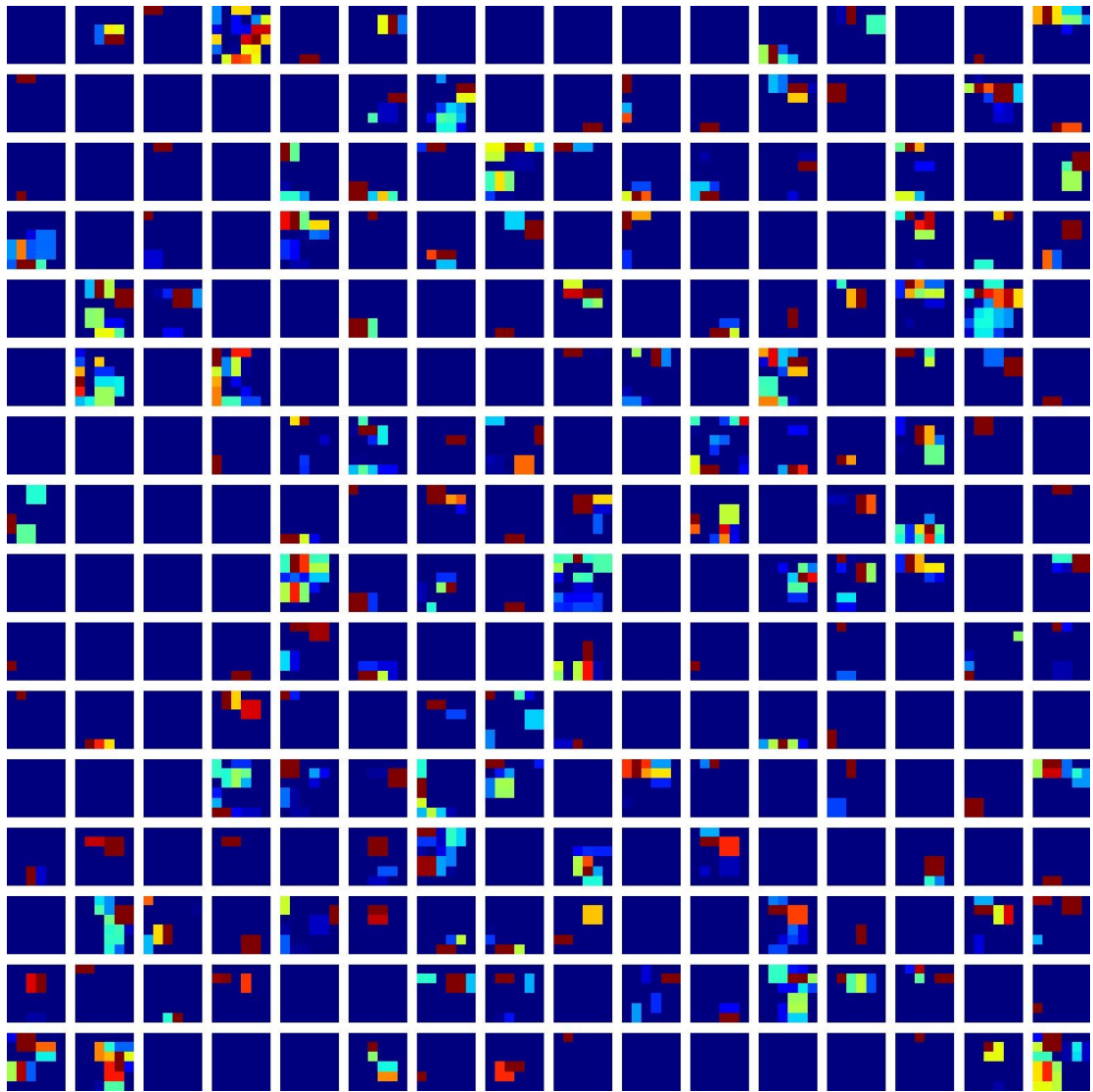








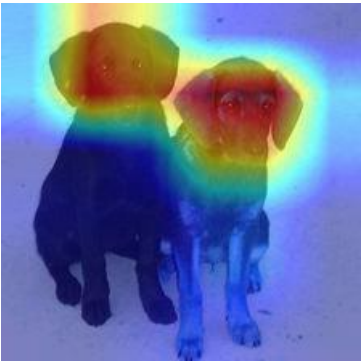


图 8: GradCAM_dog_feature_avgpool

表 2: 第一组热力图

#	原始输入	热力图 (class=0)	热力图 (class=1)
1			
		图 9: GradCAM_both_0_avgpool	图 10: GradCAM_both_1_avgpool
2			
		图 11: GradCAM_cat_0_avgpool	图 12: GradCAM_cat_1_avgpool
3			
		图 13: GradCAM_dog_0_avgpool	图 14: GradCAM_dog_1_avgpool

1.3.2 第二组实验结果

在 Grad-CAM 选取最后一个卷积层的条件下，三张输入图片的输出特征图分别如图 15~图 17 所示。对于猫和狗的分类，热力图分别如图 18~图 23 所示。

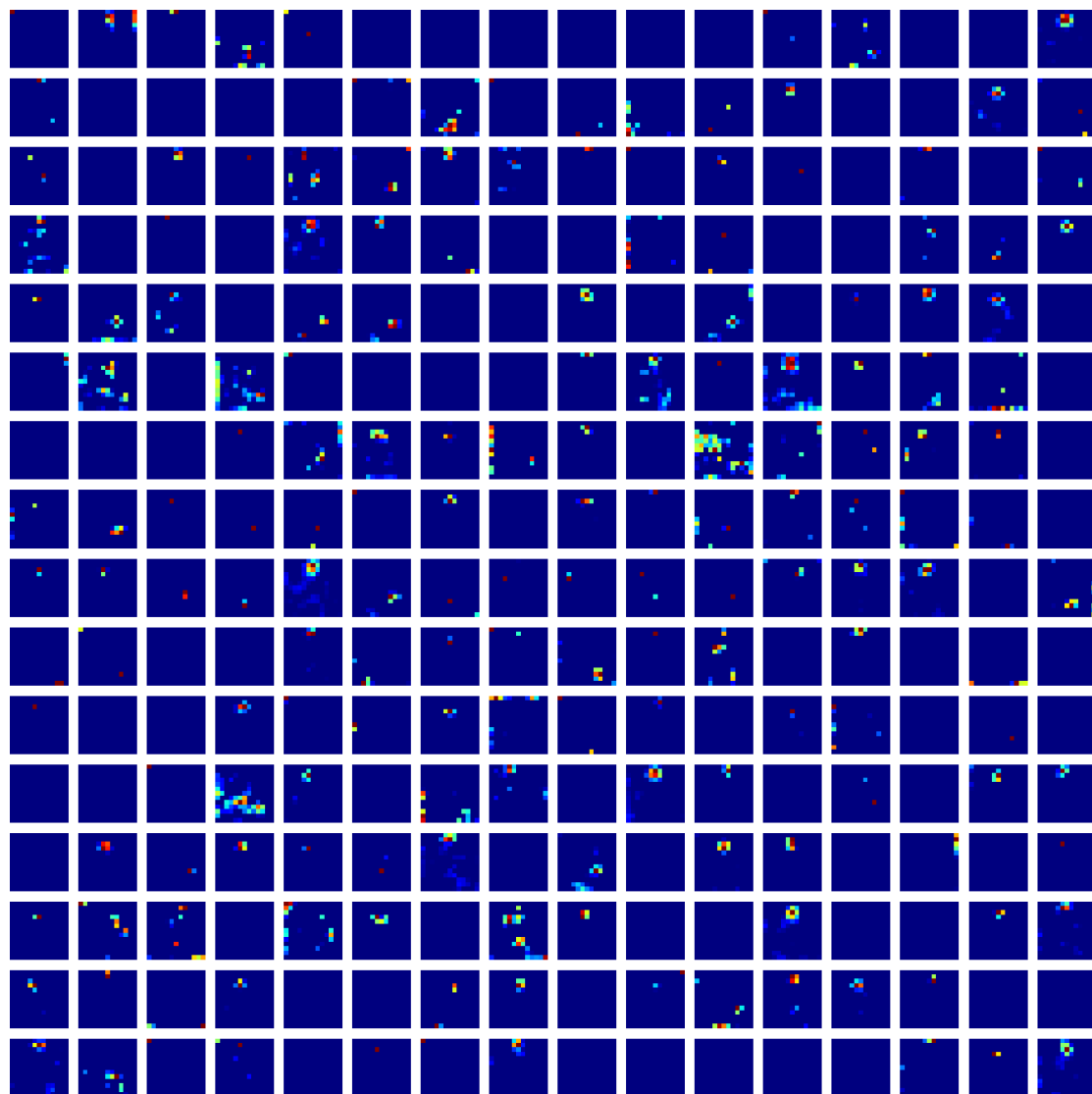


图 15: GradCAM_both_feature_lastconv

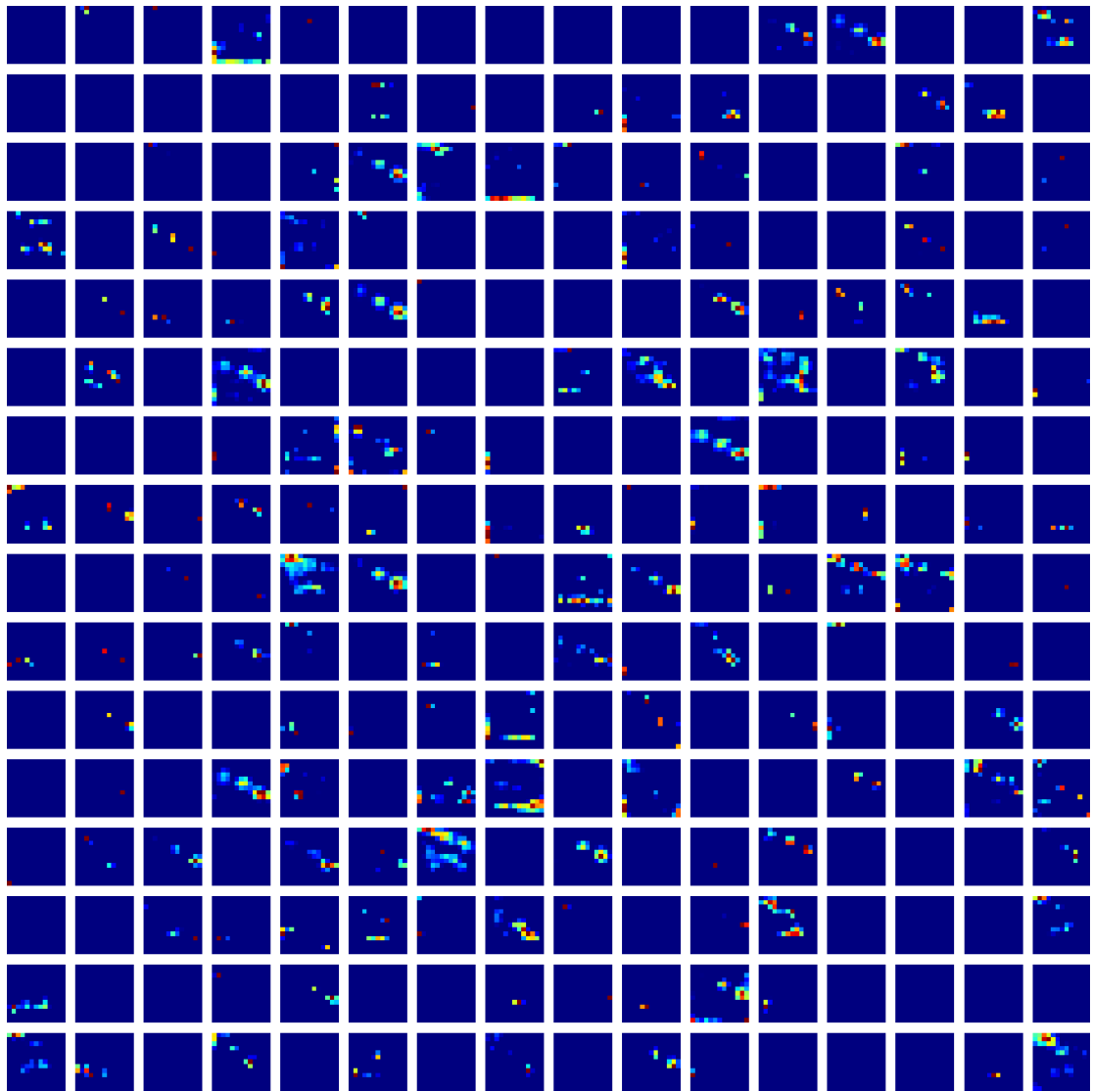


图 16: GradCAM_cat_feature_lastconv

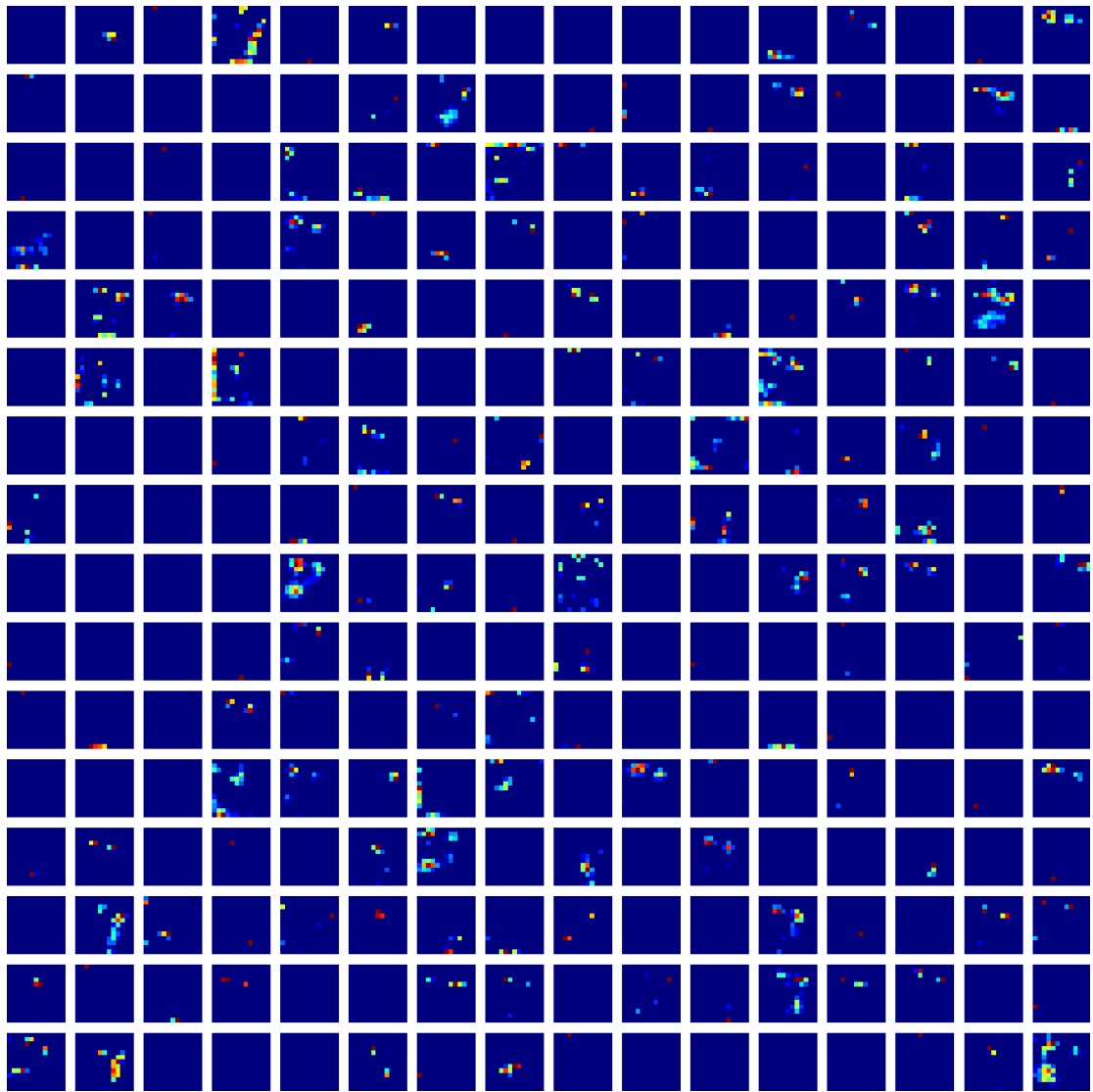







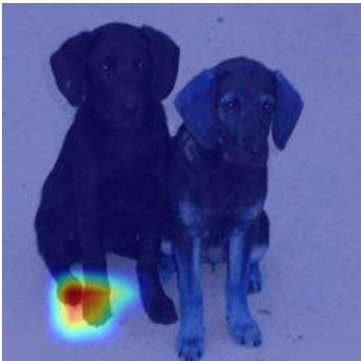
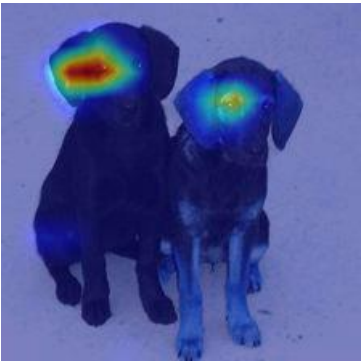


图 17: GradCAM_dog_feature_lastconv








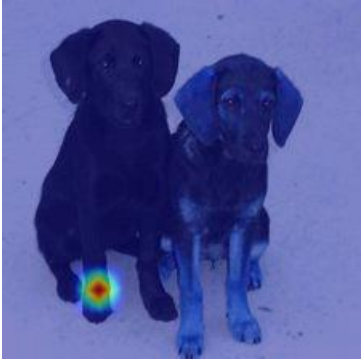

表 3：第二组热力图

#	原始输入	热力图 (class=0)	热力图 (class=1)
1			
		图 18: GradCAM_both_0_lastconv	图 19: GradCAM_both_1_ lastconv
2			
		图 20: GradCAM_cat_0_ lastconv	图 21: GradCAM_cat_1_ lastconv
3			
		图 22: GradCAM_dog_0_ lastconv	图 23: GradCAM_dog_1_ lastconv

1.3.3 第三组实验结果

在 Layer-CAM 选取最后一个卷积层的条件下，三张输入图片的输出特征图与图 15~图 17 完全一样，此处不再赘述。对于猫和狗的分类，热力图分别如图 24~图 29 所示。

表 4：第三组热力图

#	原始输入	热力图（class=0）	热力图（class=1）
1		 图 24: LayerCAM_both_0_lastconv	 图 25: LayerCAM_both_1_lastconv
2		 图 26: LayerCAM_cat_0_lastconv	 图 27: LayerCAM_cat_1_lastconv
3		 图 28: LayerCAM_dog_0_lastconv	 图 29: LayerCAM_dog_1_lastconv

1.4 结果分析

1.4.1 总体结果分析

根据以上结果可知，Grad-CAM 和 Layer-CAM 在图像的可解释性方面效果都较好。对于正确的类别，两种算法都可以画出较精确的热力图，红色区域界限明显；对于不正确的类别，算法得到的红色区域一般都落在比较奇怪的位置，说明在正常分类过程中大概率不会关注到该类别，这也从侧面解释了神经网络对图像进行分类的有效性。

1.4.2 全局平均池化层和最后一个卷积层特征图效果的对比




分别对比第一组和第二组的特征图（图 6、图 15；图 7、图 16；图 8、图 17）可知，最后一个卷积层的特征图尺寸是 13*13，保留的特征信息更多；而由于全局平均池化对各个通道都作了平均，因此特征图尺寸只有 6*6，保留的特征信息更少。

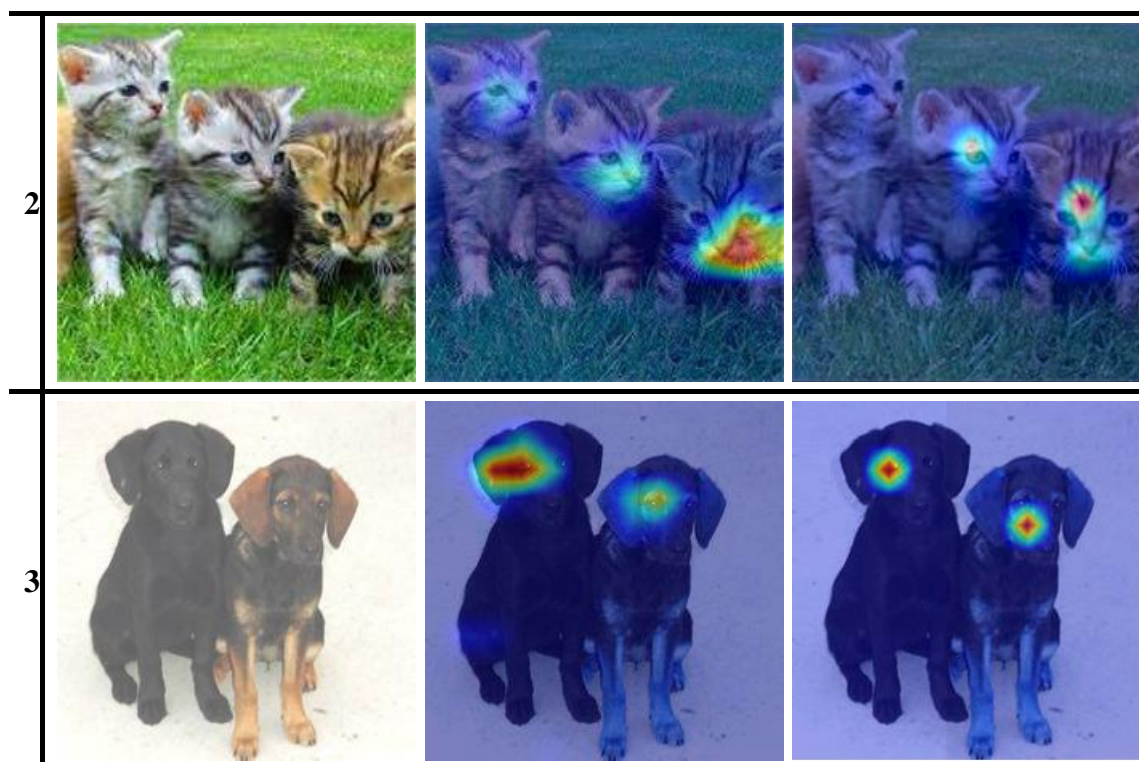
进一步，对比图 9~图 14 和图 18~图 23 可以发现基于最后一个卷积层产生的热力图更集中，覆盖范围更小；而对于全局平均池化层，因为它将预测类别的置信度分布平均到整个图像上，所以范围会更大一些。因此最后一个卷积层更适合对局部特征进行可视化。

1.4.3 Grad-CAM 和 Layer-CAM 效果的对比

把两种 CAM 算法基于最后一个卷积层（第二组和第三组）对于概率最大类别的热力图列出来，如表 5 所示。

表 5：Grad-CAM 和 Layer-CAM 热力图对比

#	原始输入	Grad-CAM	Layer-CAM
1			



观察最后两列，很显然 **Grad-CAM** 和 **Layer-CAM** 定位的目标相同、位置相近，这是符合预期的。二者的区别在于 **Layer-CAM** 的热力图更集中，并且相对于 **Grad-CAM** 的噪声更少，精确性有所提升。

1.5 参考文献

- [1] 华中科技大学计算机学院 2023 年秋季计算机视觉《第七讲 CNN 可解释性分析》
- [2] Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. <https://arxiv.org/pdf/1610.02391.pdf>
- [3] LayerCAM: Exploring Hierarchical Class Activation Maps for Localization. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9462463>

1.6 源代码

1.6.1 gradcam_avgpool.py

作用：基于全局平均池化层，使用 Grad-CAM 算法进行图像的可解释性分析。

代码：

```
import torch
import torch.nn.functional as F
from torch.autograd import Variable
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

class GradCam:
    def __init__(self, model):
        self.model = model.eval()
        self.feature = None
        self.gradient = None
        self.adaptiveavgpool_features = []

    def save_gradient(self, grad):
        self.gradient = grad

    def save_adaptiveavgpool_features(self, feature):
        self.adaptiveavgpool_features.append(feature)

    def __call__(self, x, interested_class):
        image_size = (x.size(-1), x.size(-2))
```

```

datas = Variable(x)
heat_maps = []
for i in range(datas.size(0)):
    img = datas[i].data.cpu().numpy()
    img = img - np.min(img)
    if np.max(img) != 0:
        img = img / np.max(img)

    feature = datas[i].unsqueeze(0)
    for name, module in self.model.named_children():
        if name == 'classifier':
            feature = feature.view(feature.size(0), -1)
            feature = module(feature)
        if name == 'features':
            feature.register_hook(self.save_gradient)
            self.feature = feature
        elif name == 'avgpool':
            self.save_adaptiveavgpool_features(feature) # 第12层

    classes = F.softmax(feature)

    predicted_class = interested_class
    one_hot = torch.zeros_like(classes)
    one_hot[:, predicted_class] = 1

    self.model.zero_grad()
    classes.backward(grad=one_hot, retain_graph=True)

    weight = self.gradient.mean(dim=-1,
keepdim=True).mean(dim=-2, keepdim=True)
    mask = F.relu((weight * self.feature).sum(dim=1)).squeeze(0)
    mask = cv2.resize(mask.data.cpu().numpy(), image_size)
    mask = mask - np.min(mask)
    if np.max(mask) != 0:
        mask = mask / np.max(mask)
    heat_map = np.float32(cv2.applyColorMap(np.uint8(255 * mask),
cv2.COLORMAP_JET))
    cam = heat_map + np.float32((np.uint8(img.transpose((1, 2, 0))
* 255)))
    cam = cam - np.min(cam)
    if np.max(cam) != 0:
        cam = cam / np.max(cam)

heat_maps.append(transforms.ToTensor()(cv2.cvtColor(np.uint8(255 *

```

平均池化


```

cam), cv2.COLOR_BGR2RGB)))
    heat_maps = torch.stack(heat_maps)
    return heat_maps, self.adaptiveavgpool_features

IMAGE_NAME = 'C:\GAP\计算机视觉实验\实验四\实验四模型和测试图片 (PyTorch)
\data4\both.jpg'
SAVE_NAME = 'grad_cam_both_1_avgpool.png'
test_image =
(transforms.ToTensor()(Image.open(IMAGE_NAME))).unsqueeze(dim=0)
model = torch.load('torch_alex.pth')
grad_cam = GradCam(model)

interested_class = 1 # 选择特定类别

feature_image = grad_cam(test_image,
interested_class)[0].squeeze(dim=0)
feature_image = transforms.ToPILImage()(feature_image)
feature_image.save(SAVE_NAME)

FEATURE_SAVE_NAME = 'grad_cam_both_feature_avgpool.png'

heat_map, adaptiveavgpool_features = grad_cam(test_image,
interested_class)
fig, axs = plt.subplots(16, 16, figsize=(16, 16))
count = 0

for i in range(16):
    for j in range(16):
        feature_map =
adaptiveavgpool_features[0][0][count].detach().cpu().numpy()
        axs[i, j].imshow(feature_map, cmap='jet')
        axs[i, j].axis('off')
        count += 1
        if count >= len(adaptiveavgpool_features[0][0]):
            break

plt.tight_layout()
plt.savefig(FEATURE_SAVE_NAME, bbox_inches='tight', pad_inches=0)

print(len(adaptiveavgpool_features[0][0]))

print(model.__dict__)
for name, module in model.named_modules():
    print(name, module)

```

1.6.2 gradcam_lastconv.py

作用：基于最后一个卷积层，使用 Grad-CAM 算法进行图像的可解释性分析。

代码：

```
import torch
import torch.nn.functional as F
from torch.autograd import Variable
import cv2
import numpy as np
import matplotlib.pyplot as plt
import torch
from PIL import Image
from torchvision import transforms

class GradCam:
    def __init__(self, model):
        self.model = model.eval()
        self.feature = None
        self.gradient = None

    def save_gradient(self, grad):
        self.gradient = grad

    def __call__(self, x, interested_class):
        image_size = (x.size(-1), x.size(-2))
        datas = Variable(x)
        heat_maps = []
        for i in range(datas.size(0)):
            img = datas[i].data.cpu().numpy()
            img = img - np.min(img)
            if np.max(img) != 0:
                img = img / np.max(img)

            feature = datas[i].unsqueeze(0)
            round = 0
            for name, module in self.model.named_children():
                if name == 'classifier':
                    feature = feature.view(feature.size(0), -1)
                if name == 'features':
                    for submodule in module:
                        feature = submodule(feature)
                        round += 1
                    if round == 11:
```

```

        self.feature = feature # 最后一个卷积层

        feature.register_hook(self.save_gradient)

        classes = F.softmax(feature)
        # print(self.feature)
        # print(self.feature.shape)

        # 画出特征图片
        fig, axs = plt.subplots(16, 16, figsize=(16, 16))
        count = 0

        for i in range(16):
            for j in range(16):
                feature_map = self.feature.detach().cpu().numpy()[0,
count]

                axs[i, j].imshow(feature_map, cmap='jet')
                axs[i, j].axis('off')
                count += 1
                if count >= self.feature.shape[1]:
                    break

        FEATURE_SAVE_NAME = 'grad_cam_dog_feature_lastconv.png'
        plt.tight_layout()
        plt.savefig(FEATURE_SAVE_NAME, bbox_inches='tight',
pad_inches=0)
        plt.close()

        predicted_class = torch.argmax(classes, dim=1)
        print('predicted_class: ', predicted_class)

        predicted_class = interested_class
        one_hot = torch.zeros_like(classes)
        one_hot[:, predicted_class] = 1

        self.model.zero_grad()
        classes.backward(gradient=one_hot, retain_graph=True)

        weight = self.gradient.mean(dim=-1,
keepdim=True).mean(dim=-2, keepdim=True)
        mask = F.relu((weight * self.feature).sum(dim=1)).squeeze(0)
        mask = cv2.resize(mask.data.cpu().numpy(), image_size)
        mask = mask - np.min(mask)
        if np.max(mask) != 0:

```

```

        mask = mask / np.max(mask)
        heat_map = np.float32(cv2.applyColorMap(np.uint8(255 * mask),
cv2.COLORMAP_JET))
        cam = heat_map + np.float32((np.uint8(img.transpose((1, 2, 0))
* 255)))
        cam = cam - np.min(cam)
        if np.max(cam) != 0:
            cam = cam / np.max(cam)

heat_maps.append(transforms.ToTensor()(cv2.cvtColor(np.uint8(255 *
cam), cv2.COLOR_BGR2RGB)))
        heat_maps = torch.stack(heat_maps)
        return heat_maps

IMAGE_NAME = 'C:\GAP\计算机视觉实验\实验四\实验四模型和测试图片 (PyTorch)
\data4\dog.jpg'
SAVE_NAME = 'grad_cam_dog_1_lastconv.png'
test_image =
(transforms.ToTensor()(Image.open(IMAGE_NAME))).unsqueeze(dim=0)
model = torch.load('torch_alex.pth')
grad_cam = GradCam(model)

interested_class = 1873 # 选择特定类别
# 4754 or 2507 or 1873

feature_image = grad_cam(test_image,
interested_class)[0].squeeze(dim=0)
feature_image = transforms.ToPILImage()(feature_image)
feature_image.save(SAVE_NAME)

print(model.__dict__)
for name, module in model.named_modules():
    print(name, module)

```

1.6.3 layercam.py

作用：基于最后一个卷积层，使用 Layer-CAM 算法进行图像的可解释性分析。

代码：

```

import torch
import torch.nn.functional as F
from torch.autograd import Variable
import cv2

```

```

import numpy as np
import matplotlib.pyplot as plt
from torchvision import transforms
from PIL import Image

class LayerCam:
    def __init__(self, model):
        self.model = model.eval()
        self.feature = None
        self.gradient = None
        self.lastconv_features = []

    def save_gradient(self, grad):
        self.gradient = grad

    def __call__(self, x):
        image_size = (x.size(-1), x.size(-2))
        datas = Variable(x)
        heat_maps = []
        for i in range(datas.size(0)):
            img = datas[i].data.cpu().numpy()
            img = img - np.min(img)
            if np.max(img) != 0:
                img = img / np.max(img)

            feature = datas[i].unsqueeze(0)
            round = 0
            for name, module in self.model.named_children():
                if name == 'classifier':
                    feature = feature.view(feature.size(0), -1)
                if name == 'features':
                    for submodule in module:
                        feature = submodule(feature)
                        round += 1
                    if round == 11:
                        self.feature = feature # 最后一个卷积层
                    if round == 12:
                        break
                if name == 'features' or 'classifier':
                    feature.register_hook(self.save_gradient)
            classes = F.softmax(feature)
            # print(self.feature)
            # print(self.feature.shape)

```

```

# 画出特征图片
fig, axs = plt.subplots(16, 16, figsize=(16, 16))
count = 0

for i in range(16):
    for j in range(16):
        feature_map = self.feature.detach().cpu().numpy()[0,
count]

        axs[i, j].imshow(feature_map, cmap='jet')
        axs[i, j].axis('off')
        count += 1
        if count >= self.feature.shape[1]:
            break

plt.tight_layout()
plt.savefig(FEATURE_SAVE_NAME, bbox_inches='tight',
pad_inches=0)
plt.close()

# one_hot, _ = classes.max(dim=-1)
# self.model.zero_grad()
# one_hot.backward()
# predicted_class = torch.argmax(classes, dim=1)

k = 100 # 概率最大
topk_values, topk_indices = torch.topk(classes, k)

for i in range(len(classes)):
    print(f"Top {k} classes for sample {i}:")
    for j in range(k):
        class_index = topk_indices[i, j]
        class_prob = topk_values[i, j]
        print(f"Class index: {class_index}, Probability:
{class_prob}")

predicted_class = 1992
one_hot = torch.zeros_like(classes)
one_hot[:, predicted_class] = 1
self.model.zero_grad()
classes.backward(gradient=one_hot, retain_graph=True)

print('predicted_class: ', predicted_class)

# print('feature_map.shape: ', feature_map.shape)

```

```

# print('self.feature.shape: ', self.feature.shape)
# print('self.gradient.shape: ', self.gradient.shape)

weight = self.gradient * self.feature # 对梯度进行元素级乘法
print('weight.shape: ', weight.shape)
mask = F.relu((weight * self.feature).sum(dim=1)).squeeze(0)
mask = cv2.resize(mask.data.cpu().numpy(), image_size)
mask = mask - np.min(mask)
if np.max(mask) != 0:
    mask = mask / np.max(mask)
heat_map = np.float32(cv2.applyColorMap(np.uint8(255 * mask),
cv2.COLORMAP_JET))
cam = heat_map + np.float32((np.uint8(img.transpose((1, 2, 0))
* 255)))
cam = cam - np.min(cam)
if np.max(cam) != 0:
    cam = cam / np.max(cam)

heat_maps.append(transforms.ToTensor()(cv2.cvtColor(np.uint8(255 *
cam), cv2.COLOR_BGR2RGB)))
heat_maps = torch.stack(heat_maps)
return heat_maps, self.lastconv_features

IMAGE_NAME = 'C:/GAP/计算机视觉实验/实验四/实验四模型和测试图片 (PyTorch)
/data4/dog.jpg'
SAVE_NAME = 'layer_cam_dog_0_lastconv.png'
FEATURE_SAVE_NAME = 'layer_cam_dog_feature.png'

test_image =
transforms.ToTensor()(Image.open(IMAGE_NAME)).unsqueeze(dim=0)
model = torch.load('torch_alex.pth')
layer_cam = LayerCam(model)
explainable_image = layer_cam(test_image)[0].squeeze(dim=0)
explainable_image = transforms.ToPILImage()(explainable_image)
explainable_image.save(SAVE_NAME)

```